P2 Digital Electronics

Lecture 4: Binary arithmetic

Mark Cannon

mark.cannon@eng.ox.ac.uk

Trinity Term 2026

Overview of lectures

1. Logical functions and logic gates
2. Low level logic design
3. Binary number representation
4. **Binary arithmetic**
5. Integration of digital logic components
6. Memory and sequential circuits
7. Design of sequential logic
8. Data converters: analogue to digital / digital to analogue

Please send feedback, comments and corrections to mark.cannon@eng.ox.ac.uk

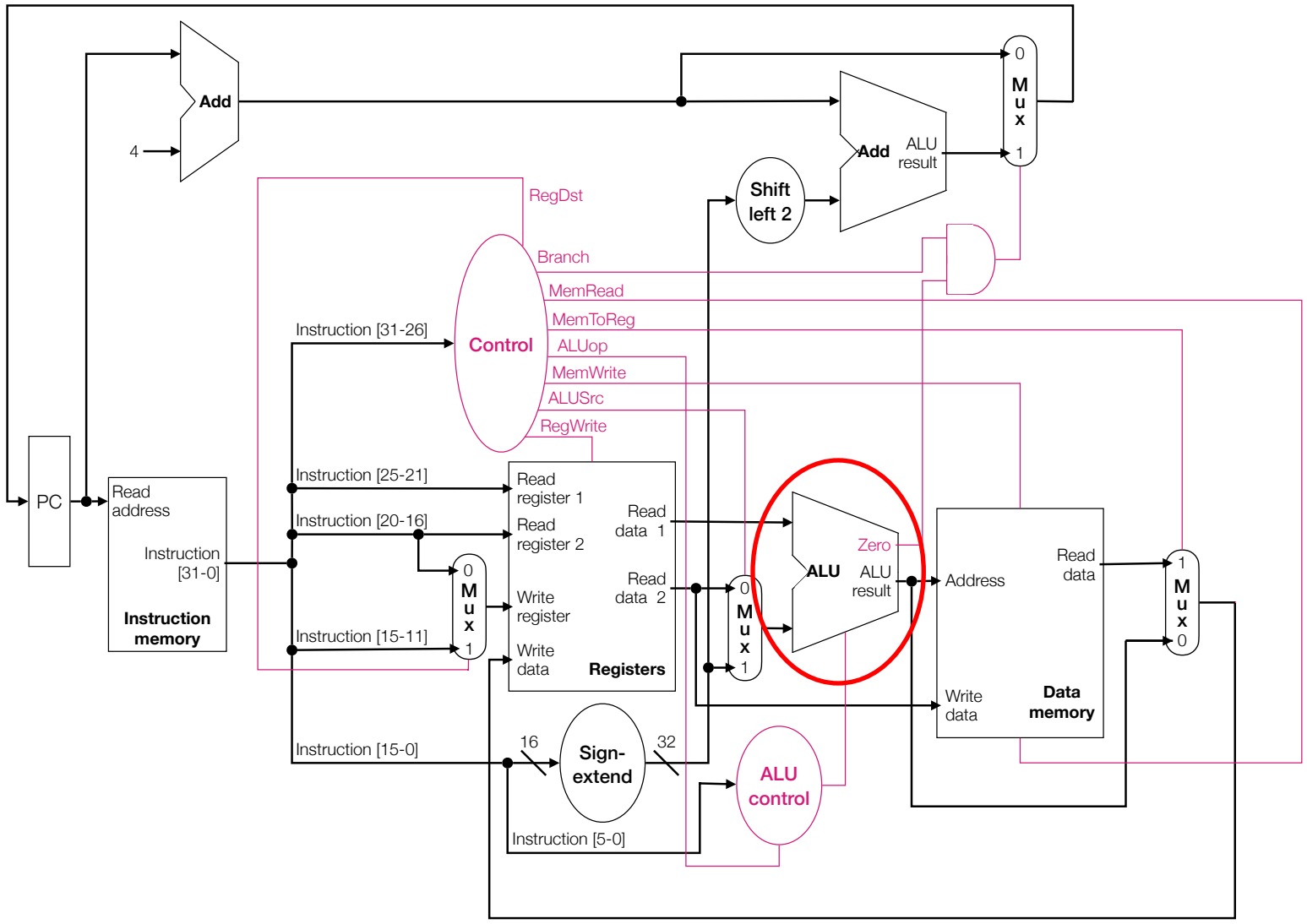
Number representations

Why do programmers confuse Halloween and Christmas?

because 31 OCT = 25 DEC

Overview of lecture 4

- Addition, half and full adders
- Ripple addition
- Subtraction
- Multiplication
- Fixed point arithmetic
- Floating point numbers



MIPS Processor – A2 Computer Architecture

Addition – the half adder

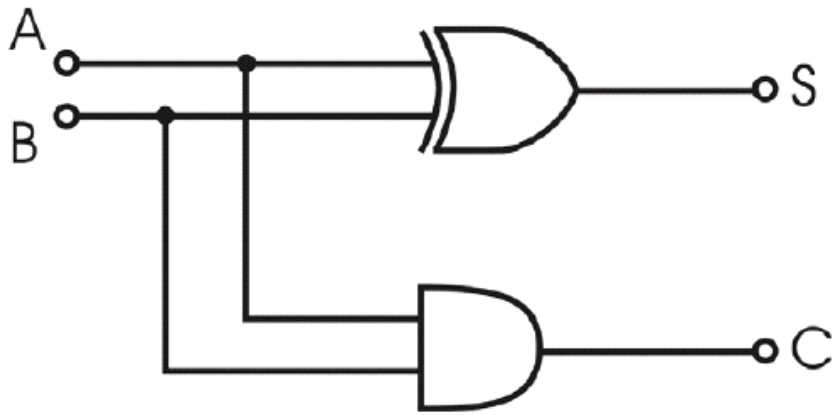
The most fundamental addition is that of two single digits (or bits)

Truth table for addition of two single bit numbers:

inputs		outputs		
A	B	Sum	Carry	Decimal
0	0	0	0	0_{10}
0	1	1	0	1_{10}
1	0	1	0	1_{10}
1	1	0	1	2_{10}

$$\begin{array}{r} 6 \\ +\ 7 \\ \hline 1\,3 \\ \hline \end{array}$$
$$\uparrow\ 1$$

A	B	XOR	AND
0	0	0	0
0	1	1	0
1	0	1	0
1	1	0	1



Full adder

If more than one bit, then there may be a carry in from the previous bit addition

```
  1 6
+ 2 7
-----
  4 3
-----
  ↖ 1
```

inputs			outputs	
A_i	B_i	C_{in}	Sum	Carry out
0	0	0	0	0
0	0	1	1	0
0	1	0	1	0
0	1	1	0	1
1	0	0	1	0
1	0	1	0	1
1	1	0	0	1
1	1	1	1	1

S_i

C_{in} \ A_iB_i	00	01	11	10
0	0	1	0	1
1	1	0	1	0

$$S_i = A_i \oplus B_i \oplus C_{\text{in}}$$

C_{out}

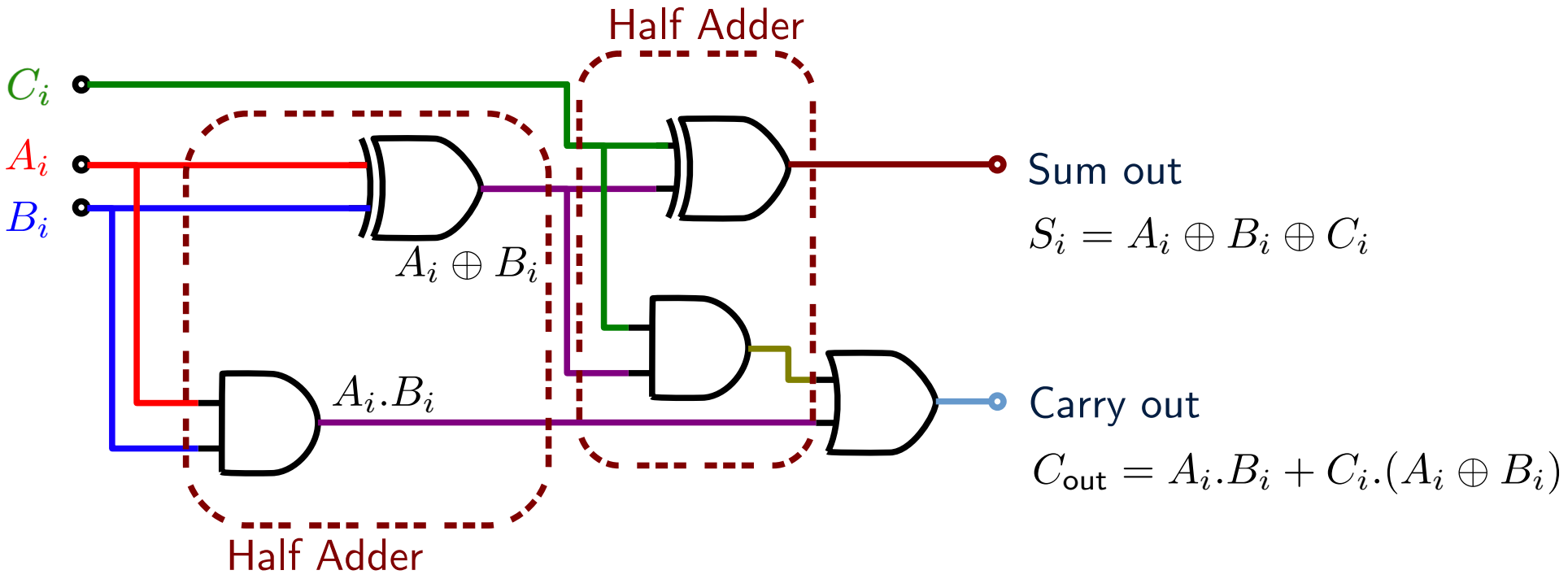
C_{in} \ A_iB_i	00	01	11	10
0	0	0	1	0
1	0	1	1	1

$C_{\text{in}}.(A_i \oplus B_i)$

$$\begin{aligned} C_{\text{out}} &= A_i.B_i + C_{\text{in}}.B_i + C_{\text{in}}.A_i \\ &= A_i.B_i + C_{\text{in}}.(A_i + B_i) \end{aligned}$$

could be XOR instead

Full adder circuit

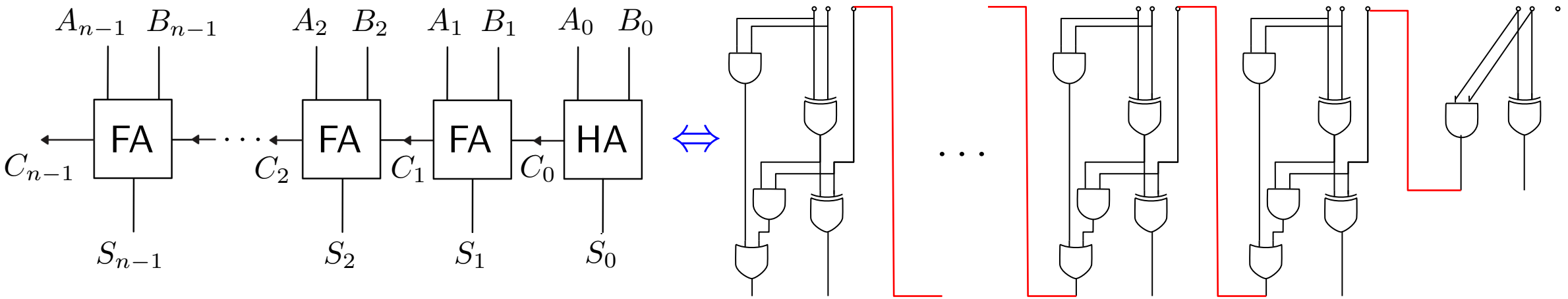


This is the building block for digital addition

It's the basis for all computation systems in use today

Ripple addition

- We want to be able to add more than 2 bits
- How can we add two n-bit numbers? $A_{n-1} \dots A_2 A_1 A_0$ $B_{n-1} \dots B_2 B_1 B_0$



- Ripple adder made with Full and Half Adders
- Sequentially add pairs of bits: n-bit addition takes n steps

Look-ahead carry

A problem with the ripple adder occurs when you try to add two numbers that come to more than you can represent

e.g. using 8-bit numbers, find $128_{10} + 154_{10}$

the answer is $128_{10} + 154_{10} = 282_{10}$

but the largest number represented by 8 bits is $255_{10} = FF_{\mathsf{H}} = 1111\,1111_2$

so $128_{10} + 154_{10} = 282_{10} = 1\,0000\,0000_2 + 0001\,1010_2$

$\Rightarrow$ need to generate a CARRY OUT to flag up the extra bit

Ideally we would like to check this in advance before using lots of time to do the ripple addition

Looking ahead

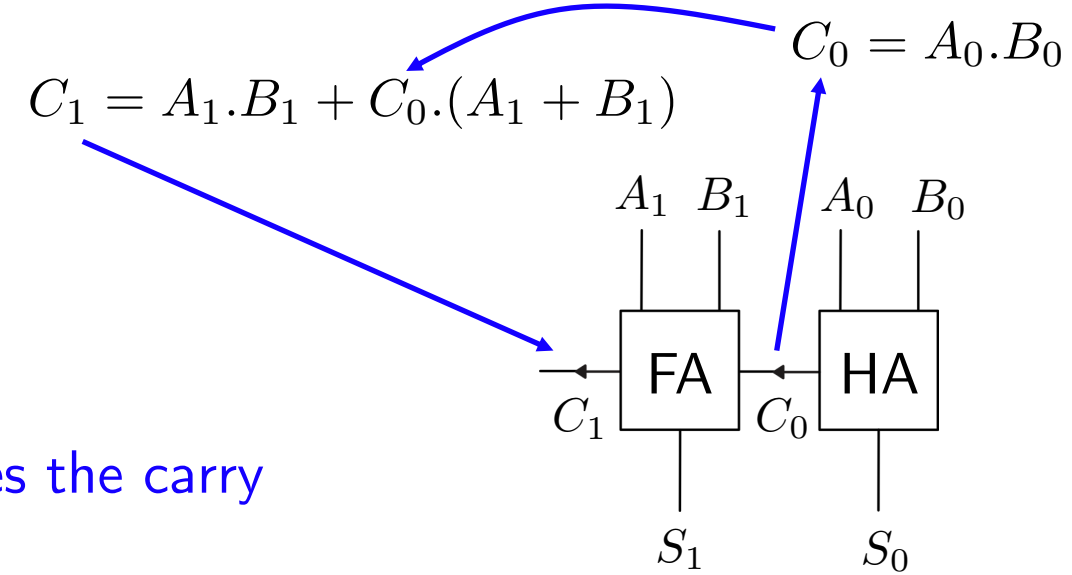
Full adder $C_{\text{out}} = A_i.B_i + C_{\text{in}}.(A_i + B_i)$

Half adder $C_{\text{out}} = A_i.B_i$

e.g. if we add two 2-bit numbers: $A_1A_0,\ B_1B_0$

substitute...

$C_1 = A_1.B_1 + (A_0.B_0).(A_1 + B_1)$



Now have a single logic function that calculates the carry out without needing to do full addition

Can be extended to more digits in a similar manner

Subtraction

We have already established that subtraction can be performed via addition using 2's complement representation

e.g. to compute $S = A - B$:

1. find the 2's complement representation of $-B$
2. compute $A + (-B)$

- How do we build a circuit to implement subtraction?
- How can we use the same circuit for performing both addition and subtraction?

4-bit adder/subtractor

To compute $S = A - B$:

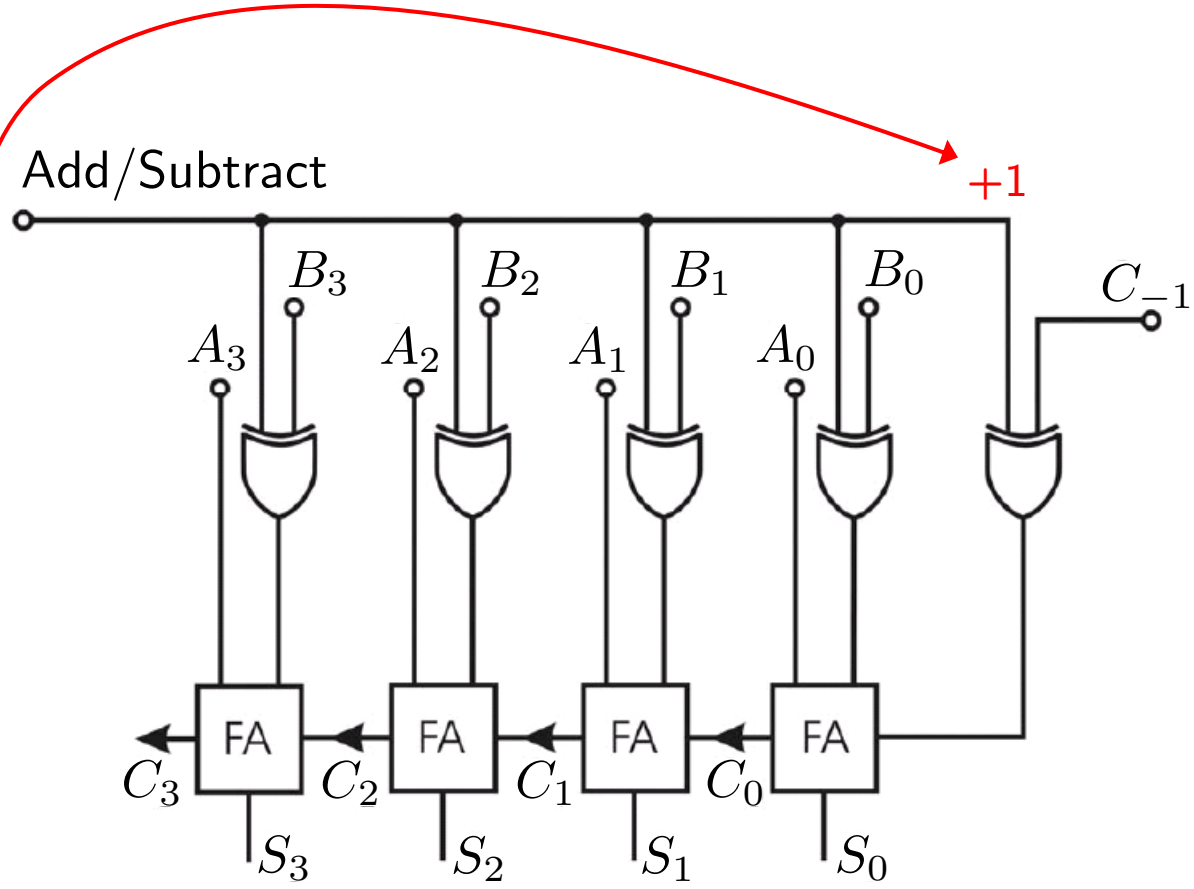
1. find the 2's complement representation of $-B$:
   (a). $B_{n-1} \ldots B_1 B_0 \rightarrow \overline{B}_{n-1} \ldots \overline{B}_1 \overline{B}_0$

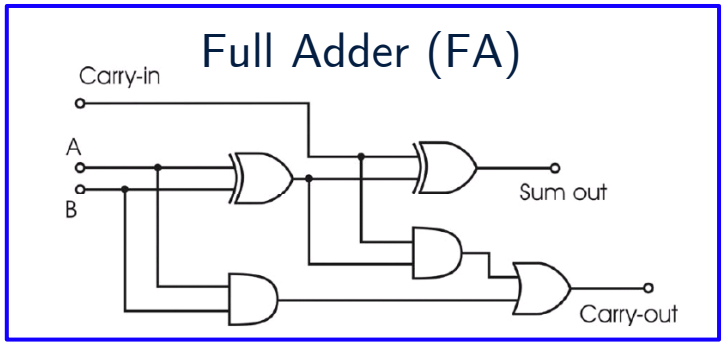
	Add/ Subtract	B_i	XOR
Add	0	0	0
	0	1	1
Subtract	1	0	1
	1	1	0

   (b). add 1

NB C_{-1} must be 0 for subtraction

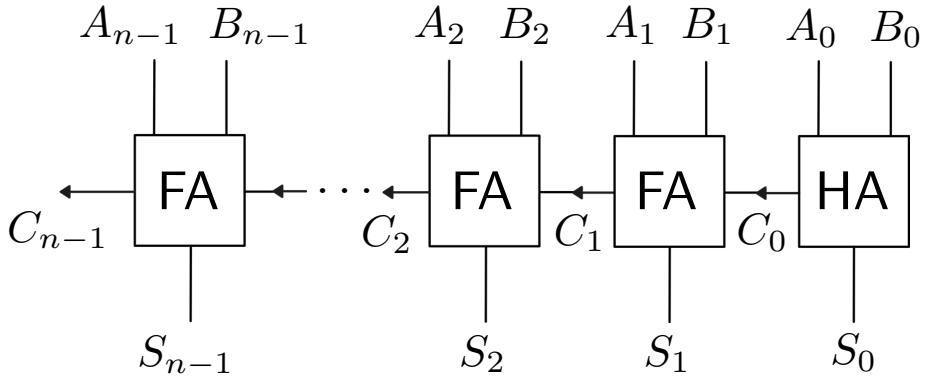
2. compute $A + (-B)$



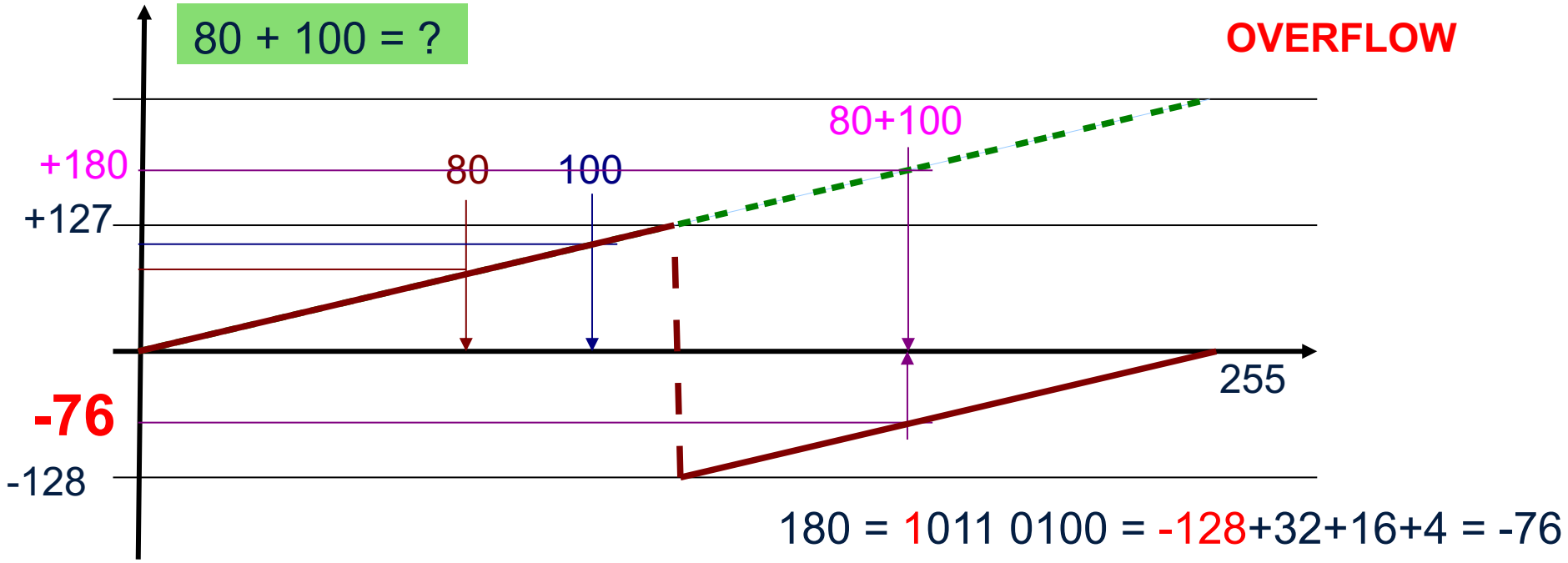


Carry and Overflow flags

- Carry flag (a single bit output) indicates sum of two binary numbers will generate a carry out



- If using 2's complement there's an extra problem...



How long is a word?

- "Binary digit" became the word "bit"
- Standard sized grouping of bits was termed "byte"
  -- Today, this is almost universally 8 bits
- A "word" is the standard data size for a particular computer architecture
  -- Today usually 32 or 64 bits = 4 or 8 bytes
- A "syllable" was used in the past for byte/word
- A "nybble" is (naturally) half a byte

How many bytes in a kilobyte?

- Metric approach (SI) $1\,\mathrm{kB} = 1000$ bytes
- "Binary" approach uses $1\,\mathrm{kB} = 2^{10} = 1024$ bytes
- Both are in use, so be careful which is relevant
- So, 1 terabyte (1 tB) could be $1\text{x}10^{12}$ bytes or $1.1\text{x}10^{12}$ bytes

Multiplication

Decimal

```
  1357
x   11
------
  1357
 13570
------
 14927
------
```

Binary

```
   1011  <-- A
x    11  <-- B
-------
   1011
  10110  <-- A shifted one place to left
-------
 100001  <-- A + shifted version of itself
-------
```

For each non zero bit in B, add a shifted copy of A to the result

```
      1011
    x 1111
-----------
      1011
     10110
    101100
   1011000
-----------
  10100101
-----------
```

```
      1011
    x 1101
-----------
      1011
     00000
    101100
   1011000
-----------
  10000111
-----------
```

MAX NUMBER OF BITS REQUIRED $= 2n$

Fixed point arithmetic

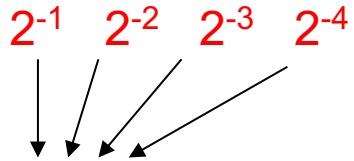
So far, we have considered only integer arithmetic – now consider numbers with a decimal (or binary) point:

Decimal

$$3.1415_{10} = 3\times10^{0} + 1\times10^{-1} + 4\times10^{-2} + 1\times10^{-3} + 5\times10^{-4}$$

$$= 3\times1 + \frac{1}{10} + \frac{4}{100} + \frac{1}{1000} + \frac{5}{10000}$$

Binary



$$101.1011_{2} = 1\times4 + 1\times1 + \frac{1}{2} + \frac{1}{8} + \frac{1}{16} = 5.6875$$

Fixed point numbers

– have a fixed number of bits before and after the ~~decimal~~ binary point

– e.g. `0000.0001` smallest number possible = 1/16

`1111.1111` largest number possible = 2^5 - 1/16 = 15.9375

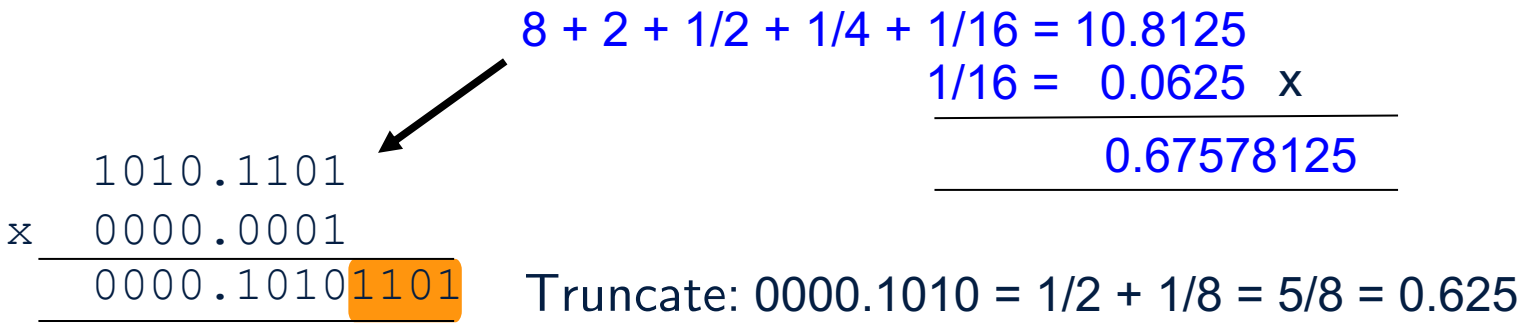
Fixed point addition

```
  1010.1101
+ 0101.0011
-----------
 10000.0000
```

Just like normal addition - add bitwise as usual

Possibility of carry out being generated

Fixed point multiplication



We can no longer accurately represent the answer in fixed point representation
- this phenomenon is known as "underflow"

Q: How can we get around this problem?

A: Use floating point representation - allow number of digits before and after the binary point to change according to requirements

Floating point numbers

We often use "scientific notation" for numbers

e.g. $c = 2.9979 \times 10^8 \simeq 299\ 792\ 458\ \text{ms}^{-1}$

-- The decimal point location is determined by the exponent

-- The value is determined by the mantissa

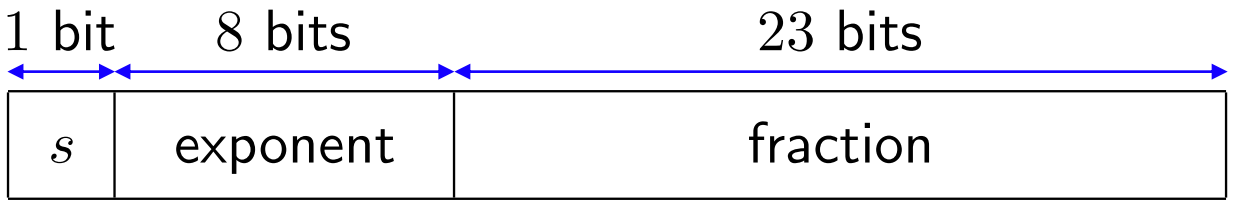
-- We can do the same in binary:

e.g. $1011.1010 = 1.0111010 \times 2^3$

-- Useful fact: all binary numbers start with 1

Generalised binary floating point

IEEE standard 32-bit floating point number: $\pm 1.XXXXXXXXX \times 2^f$



sign bit:

$$s = \begin{cases} 1 & -\text{ve} \\ 0 & +\text{ve} \end{cases}$$

exponent

shifted binary:
$f = \text{exponent} - 127$

fraction

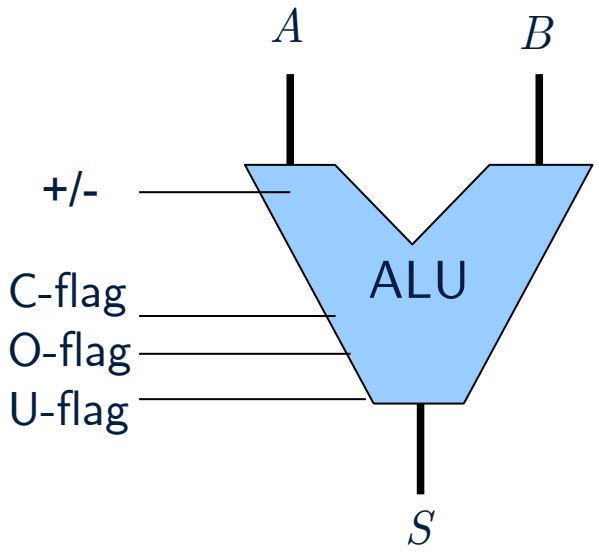
contains only the bits after the binary point:
mantissa $= 1.XXXXXXXXX$

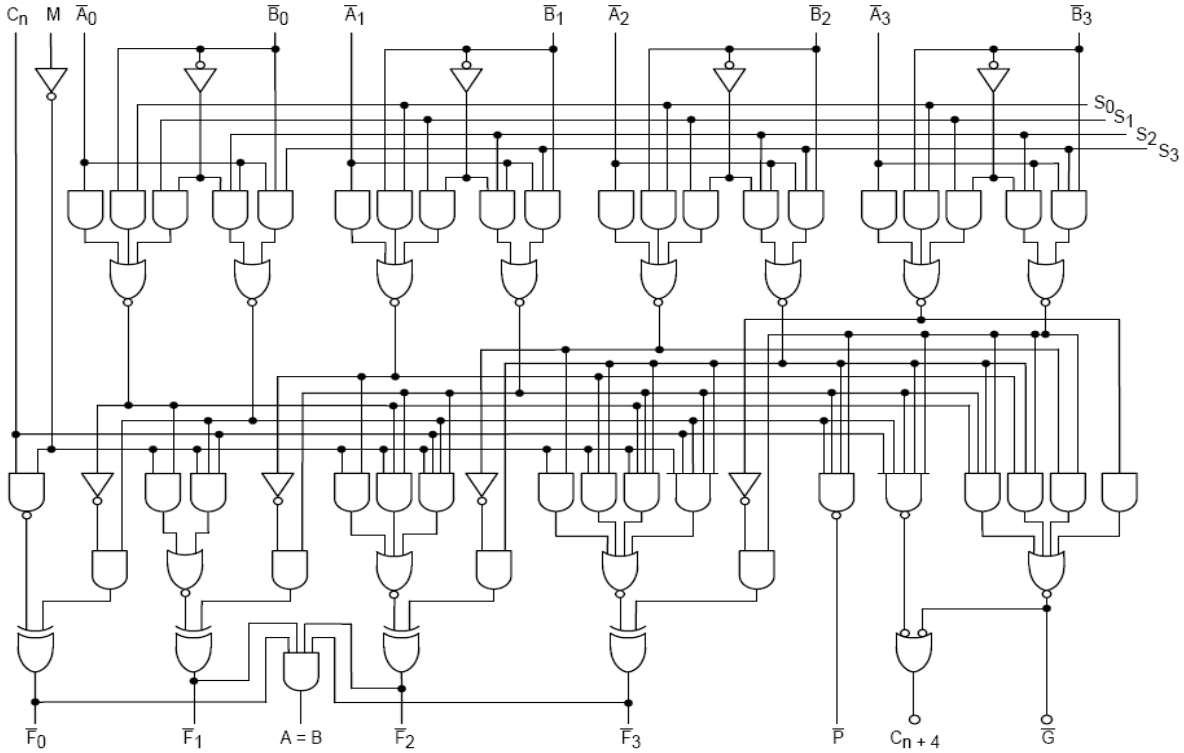
$$1\,0111\,0101\,101\,0100\,1001\,0111\,1101\,0101 = -1 \times 2^{117-127} \times 1.101\,0100\,1001\,0111\,1101\,0101_2$$
$$= -1 \times 2^{-10} \times 1.66088354587554931641_{10}$$
$$= (-1.621957 \times 10^{-3})_{10} \qquad (6\text{ d.p.})$$

Arithmetic logic unit (ALU)

The calculating part of a computer processor

- Contains a full ripple adder (e.g for 8 bits, etc.)
- Driven by control lines +/-
- Two input numbers (e.g. 8 bit)
- One output (e.g. 8 bit)
- Flag-lines indicate fault conditions:
  - -- Carry-out
  - -- Overflow
  - -- Underflow





Overview of lectures

1. Logical functions and logic gates
2. Low level logic design
3. Binary number representation
4. **Binary arithmetic**
5. Integration of digital logic components
6. Memory and sequential circuits
7. Design of sequential logic
8. Data converters: analogue to digital / digital to analogue

Please send feedback, comments and corrections to mark.cannon@eng.ox.ac.uk